Approche hybride pour SAT

Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure

Approche hybride pour SAT*

Gilles Audemard Jean-Marie Lagniez Bertrand Mazure

Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens

{audemard,lagniez,mazure}@cril.fr

Résumé

Cet article introduit SATHYS *(SAT Hybrid Solver), une nouvelle approche hybride pour le problème de la satisfiabilité propositionnelle (SAT). Cette approche combine la recherche locale et un solveur CDCL (« Conflict Driven Clause Learning »). Chaque fois qu'un minimum local est atteint par la recherche locale, un ensemble de variables est fixé à l'aide du solveur CDCL. Pour des instances satisfiables le fait de fixer des variables revient à les considérer comme tabou. Dans le cas où l'instance n'admet pas de modèle, le solveur CDCL focalisera sa recherche sur un sous-ensemble de formules insatisfiables (*MUS*). Les expérimentations fournissent de bons résultats sur un large panel d'instances issues des dernières compétitions SAT.*

Mots Clef

Recherche locale, CDCL, SAT, Hybridation.

Abstract

In this article, a novel hybrid approach for propositional satisfiability problem, called SATHYS *(*SAT HYbrid Solver*), is introduced. It combines local search and conflict driven clause learning (CDCL) schemes. At each time that a local minima is reached by the local search algorithm, a CDCL is launched. For satisfiable instances, the CDCL component behaves like a tabu list, whereas for unsatisfiable ones, it tries to focus on minimum unsatisfiable subformula (MUS). It is shown in the experimental validation that this approach obtains good results on many classes of SAT instances issued from the last SAT competitions.*

Keywords

Local search, CDCL, SAT, Hybridization.

1 Introduction

Le problème SAT est le problème de décision qui consiste à vérifier si une formule booléenne, souvent représentée sous Forme Normale Conjonctive (CNF), admet ou non une affectation de ses variables qui associe la valeur vrai à la formule. Ce problème est très important en théorie de la complexité où il représente le problème NP-complet de référence et possède de nombreuses applications en vérification formelle, planification, raisonnement non monotone... Ces deux dernières décennies, beaucoup d'approches ont été proposées pour résoudre ce problème. Elles peuvent se décomposer en de deux catégories : les méthodes complètes et incomplètes. Les algorithmes incomplets sont généralement basés sur la recherche locale [29, 28, 17] alors que les approches complètes sont généralement une amélioration de la procédure bien connue de Davis-Putnam-Loveland-Logemann, communément appelée DPLL [4]. Parmi elles, on trouve les solveurs de type CDCL (« *Conflict Driven Clause Learning* ») [27, 6], aujourd'hui appelés solveurs SAT modernes. Ces derniers sont capables de résoudre des problèmes contenant un grand nombre de variables et de clauses. Les deux approches présentées précédemment sont complémentaires. En effet, les solveurs SAT modernes sont particulièrement efficaces sur des instances issues d'applications industrielles alors que la recherche locale fournit de bons résultats sur des problèmes générés aléatoirement.

Cette complémentarité permet de supposer qu'hybrider la recherche locale (RL) avec un solveur SAT moderne (CDCL) constitue une piste intéressante. Notons que, dès 1997, Selman *et al.* ont fait de la réalisation d'un solveur hybride efficace un des dix challenges proposés à la communauté SAT [30]. Ce challenge était toujours d'actualité lors du bilan qu'ils ont établi en 2003 [22]. Le but de cette hybridation est d'exploiter les qualités des différentes approches, une hybridation parfaite étant une méthode meilleure que les deux approches, RL et CDCL, prises séparément. Un certain nombre de tentatives ont été effectuées cette dernière décennie. Celles-ci seront introduites dans la section 3.

Dans cet article, nous proposons une nouvelle hybridation entre la recherche locale et un solveur CDCL, nommée SATHYS (« *SAT Hybrid Solver* »). Notre approche a pour base la recherche locale. Lorsqu'un minimum local est atteint et

*supporté par le projet ANR UNLOC

qu'un certain nombre de conditions sont remplies (voir section 6), la partie CDCL fixe certaines variables afin d'obtenir des comportements différents suivant la satisfiabilité de la formule. Dans le cas d'une formule satisfiable, le solveur CDCL simule une liste tabou [11]. À l'opposé, dans le cas d'une formule insatisfiable, il essaie de focaliser la recherche sur les sous-formules insatisfiables les plus petites possibles (idéalement des MUS, « *Minimum Unsatisfiable Sub-Formula* ») [7, 13, 14].

Le reste de l'article est organisé de la manière suivante. Les sections 2 et 3 introduisent différentes notions et notations utiles à la compréhension de l'article. La section 4 discute des différentes approches hybrides existantes. Ensuite, l'intuition se trouvant derrière notre méthode est présentée (section 5). Dans la section 6, nous donnons les détails et l'algorithme SATHYS. Enfin les deux dernières sections sont consacrées aux expérimentations (section 7) et à la conclusion (section 8).

2 Définition et notation

Dans cette section nous donnons quelques définitions et notations. Soit $\mathcal{V} = \{x_1, ..., x_n\}$ un ensemble de variables propositionnelles, un littéral ℓ est soit une variable mise sous forme positive x_i ou négative $\neg x_i$. Le littéral opposé de ℓ est noté $\bar{\ell}$. Une clause $c_i = (\ell_1 \vee \cdots \vee \ell_{n_i})$ est une disjonction de littéraux. Une clause unitaire est une clause avec un seul littéral, appelé littéral unitaire. Une formule Σ est sous forme normale conjonctive (CNF) si elle est composée d'une conjonction de clauses $\Sigma = (c_1 \wedge \cdots \wedge c_m)$. L'ensemble des variables appartenant à Σ est noté $\mathcal{V}_\Sigma$. Une interprétation $\mathcal{I}$ d'une formule booléenne Σ associe une valeur de vérité $\mathcal{I}(x)$ aux variables $x \in \mathcal{V}_\Sigma$. L'interprétation $\mathcal{I}$ est dite *complète* (notée $\mathcal{I}_c$) si elle associe une valeur de vérité à chaque variable $x \in \mathcal{V}_\Sigma$, sinon elle est dite *partielle* (notée $\mathcal{I}_p$). Une clause, une formule CNF et une interprétation peuvent être représentées par des ensembles respectivement de littéraux, de clauses et de littéraux. Une clause c est satisfaite sous une une interprétation $\mathcal{I}$ si et seulement si $\mathcal{I} \cap c \neq \emptyset$. Un *modèle* d'une formule Σ, noté $\mathcal{I} \models \Sigma$, est une interprétation $\mathcal{I}$ qui satisfait la formule Σ, c'est-à-dire qu'elle satisfait chaque clause de Σ. À l'opposé, une interprétation $\mathcal{I}$ est appelée un *nogood* de Σ, noté $\mathcal{I} \not\models \Sigma$, si elle falsifie au moins une clause de Σ. Nous pouvons alors définir le problème de décision SAT de la manière suivante : étant donnée une formule CNF Σ, la question est de déterminer si Σ admet ou non un modèle.

Les notations suivantes sont utilisées dans le reste de l'article :

- soit L un ensemble de littéraux, $\bar{L} = \{\bar{\ell} | \ell \in L\}$ désigne l'ensemble des littéraux complémentaires de L ;
- $\Sigma_{|\ell}$ dénote la formule simplifiée par l'affectation du littéral ℓ à vrai, c'est-à-dire $\Sigma_{|\ell} = \{c \setminus \{\bar{\ell}\} | c \in \Sigma \text{ et } \ell \notin c\}$. Cette notation peut être étendue aux interprétations. Soit $\mathcal{I} = \{\ell_1, \ldots, \ell_n\}$ une interprétation, alors $\Sigma_{|\mathcal{I}} = ((\Sigma_{|\ell_1}) \ldots)_{|\ell_n}$;
- Σ^* dénote la formule Σ simplifiée par propagation unitaire ;
- $\models_*$ dénote la déduction logique par propagation unitaire : $\Sigma \models_* \ell$ signifie que le littéral l est déduit par propagation unitaire de Σ, c'est-à-dire $(\Sigma \wedge \bar{\ell})^* \models \bot$. $\Sigma \models_* \bot$ dénote le fait que la formule est insatisfiable par propagation unitaire.
- considérons Σ une formule CNF et $\mathcal{I}_c$ une interprétation complète. On dira que ℓ satisfait (resp. falsifie) une clause $\beta \in \Sigma$ si $\ell \in \mathcal{I}_c \cap \beta$ (resp. $\ell \in \mathcal{I}_c \cap \bar{\beta}$). Nous notons $\mathcal{L}^+_{\mathcal{I}_c}(\beta)$ (resp. $\mathcal{L}^-_{\mathcal{I}_c}(\beta)$) l'ensemble des littéraux satisfaits (resp. falsifiés) dans la clause β par $\mathcal{I}_c$. On note aussi $\mathcal{C}^-_{\mathcal{I}_c}(\Sigma)$ l'ensemble des clauses de Σ falsifiées par l'interprétation $\mathcal{I}_c$.

Les définitions suivantes sont tirées de [12].

Définition 1 *Une clause β est dite unisatisfaite par le littéral z pour une interprétation $\mathcal{I}_c$ si $\mathcal{L}^+_{\mathcal{I}_c}(\beta) = \{z\}$.*

Définition 2 *Soit $\mathcal{I}_c$ une interprétation complète. Une clause α est critique relativement à $\mathcal{I}_c$ si $|\mathcal{L}^+_{\mathcal{I}_c}(\alpha)| = 0$ et $\forall \ell \in \alpha, \exists \alpha' \in \Sigma$ tel que $\bar{\ell} \in \alpha'$ et $\mathcal{L}^+_{\mathcal{I}_c}(\alpha') = \{\bar{\ell}\}$. Les clauses α' sont dites liées à α.*

Exemple 2.1 *Soient $\Sigma = (\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c}) \wedge (c \vee \bar{a})$ une formule CNF et $\mathcal{I}_c = \{a, b, c\}$ une interprétation. La clause $\alpha_1 = (\bar{a} \vee \bar{b} \vee \bar{c})$ est critique. Les autres clauses sont liées à α_1 en fonction de $\mathcal{I}_c$.*

3 Notions Préliminaires

3.1 Algorithme de recherche locale

Le schéma de recherche locale est relativement simple. Il consiste à se déplacer de manière stochastique sur l'ensemble des interprétations complètes de la formule à résoudre. À chaque étape on essaie, par l'inversion de la valeur de vérité d'une variable, de réduire le nombre de clauses falsifiées (phase de descente). La prochaine interprétation est choisie parmi l'ensemble des interprétations voisines (c'est-à-dire différant uniquement sur la valeur d'une seule variable) de l'interprétation courante. Lorsque aucune interprétation voisine ne permet de diminuer le nombre de clauses insatisfaites, on se trouve dans un minimum local. L'un des points essentiels des méthodes de recherche locale est la technique utilisée pour sortir des minima locaux. Pour plus d'informations sur les différentes méthodes de recherche locale le lecteur pourra se référer à [18].

Dans la suite nous présentons l'algorithme WSAT (Algorithme 1). Cet algorithme commence par générer une interprétation complète et va ensuite effectuer des modifications locales. Pour ce faire, une clause falsifiée par l'interprétation courante est choisie de manière aléatoire (ligne 6). S'il existe une variable appartenant à cette clause permettant de diminuer le nombre de clauses fausses (descente), on inverse (« *flip* ») sa valeur de vérité (ligne 8). Sinon, on se trouve dans un minimum local et un critère d'échappement est utilisé (ligne 10). Afin de diversifier la recherche,

Algorithme 1 : WSAT

Input : Σ une formule CNF
Output : SAT si Σ satisfiable, $UNKNOWN$ si pas de modèle trouvé

```
1  for i ← 1 to MaxTries do
2      I_c ← interprétation complète de Σ;
3      for j ← 1 to MaxFlips do
4          if I_c ⊨ Σ then
5              return SAT;
6          α ∈ Σ tel que I_c ⊭ α;
7          if ∃x ∈ α permettant une descente then
8              flipper(x)
9          else                          /* minimum local */
10             Remplacer I_c par une interprétation obtenue selon
               un critère d'échappement;
11     if I_c ⊨ Σ then
12         return SAT;
13 return UNKNOWN;
```

tous les $MaxFlips$ réparations locales, l'algorithme génére aléatoirement une nouvelle interprétation complète. Ce processus peut être répété un certain de nombre de fois ($MaxTries$) fixé au départ.

3.2 Solveur CDCL

L'algorithme 2 donne le schéma général d'un solveur de type CDCL. Pour des raisons d'espace, nous ne pouvons pas détailler toutes les composantes (fonctions) de ce solveur. Typiquement, un tel solveur peut être assimilé à une séquence de décisions et de propagation des littéraux unitaires. Chaque littéral choisi comme littéral de décision (lignes 18-20) est affecté à un niveau de décision (nd), les littéraux déduits (par propagation unitaire) de cette décision ont le même niveau. Si tous les littéraux sont affectés, alors $\mathcal{I}$ est un modèle de Σ (lignes 16-17). À chaque fois qu'un conflit est atteint par propagation (γ est la clause conflit), un *nogood* β est calculé (ligne 8) en utilisant une méthode d'analyse de conflits donnée, le plus souvent le schéma « *First UIP* » (*Unique Implication Point* [31]) et un niveau de « *backjump* » bl est calculé. À ce moment, il est possible de prouver l'inconsistance de la formule (ligne 10). Si ce n'est pas le cas, on fait un saut arrière et le niveau de décision devient égal au niveau de *backjump* (ligne 13-14). Finalement, certains solveurs CDCL forcent le redémarrage (diverses stratégies sont possibles, voir [19] par exemple) et dans ce cas, on remonte en haut de l'arbre (ligne 12).

3.3 Formules minimales insatisfiables (MUS)

Une sous-formule minimalement insatisfiable représente une plus petite cause d'incohérence d'une instance SAT en terme de nombre de clauses intervenant dans l'incohérence. Formellement, on a :

Algorithme 2 : Solveur CDCL

Input : une formule CNF Σ
Output : SAT ou UNSAT

```
1  I ← ∅ ;                          /* interprétation */
2  dl ← 0 ;                  /* niveau de décision */
3  x_c ← 0 ;                 /* nombre de conflits */
4  while (true) do
5      γ ← propagationUnitaire(Σ,I);
6      if (γ! = ∅) then
7          x_c ← x_c + 1;
8          β ← analyseConflit(Σ,I,γ);
9          bl ← calculNiveauBackjumping(β,I);
10         if (bl < 0) then return UNSAT;
11         Σ ← Σ ∪ {γ};
12         if (restart()) then bl = 0;
13         backjump(Σ,I,bl);
14         dl ← bl;
15     else
16         if (toutes les variables sont affectées) then
17             return SAT;
18         ℓ ← choisirLitéralDecision(Σ);
19         dl ← dl + 1;
20         I ← I ∪ {ℓ};
```

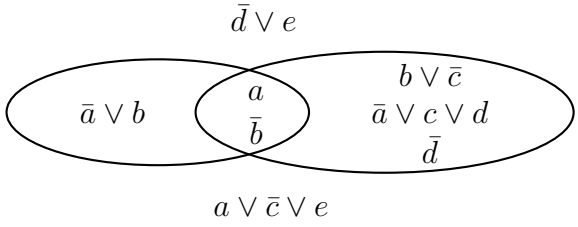

FIG. 1 – L'ensemble des MUS de Σ (exemple 3.1)

Définition 3 *Soit Σ une formule CNF. Un* MUS *Γ de Σ est un ensemble de clauses tel que :*

1. $\Gamma \subseteq \Sigma$ *;*
2. Γ *est insatisfiable ;*
3. $\forall \Delta \subset \Gamma, \Delta$ *est satisfiable.*

Exemple 3.1 *Soit $\Sigma = \{\bar{d} \vee e,\ b \vee \bar{c},\ \bar{d},\ \bar{a} \vee b,\ a,\ a \vee \bar{c} \vee e,\ \bar{a} \vee c \vee d,\ \bar{b}\}$ une formule CNF. La figure 1 représente l'ensemble des* MUS *de Σ.*

Un MUS étant insatisfiable on peut déduire la proposition suivante :

Proposition 1 *Soient Σ une formule CNF, Γ un* MUS *de Σ. Pour toute interprétation complète $\mathcal{I}_c$ construite sur $\mathcal{V}_\Sigma$, il existe $\alpha \in \Gamma$ tel que $\mathcal{I}_c \not\models \alpha$.*

De la proposition précédente, il est possible de déduire le corrolaire suivant.

Corrolaire 2 *Soient Σ une formule CNF, Γ un* MUS *de Σ. Pour toute interprétation complète $\mathcal{I}_c$ construite sur $\mathcal{V}_\Sigma$, il existe $\alpha \in \Gamma$ totalement satisfaite par $\mathcal{I}_c$.*

En effet, dans le cas contraire, l'interprétation opposée ($\bar{\mathcal{I}_c}$) serait un modèle de la formule.
De la proposition et du corrolaire précédents, il est possible de définir une mesure permettant d'estimer le degré d'appartenance d'une clause α à un MUS. Cette mesure sera notée ϵ.

Définition 4 *Soient Σ une formule CNF insatisfiable, $\mathcal{I}_c$ une interprétation complète construite sur $\mathcal{V}_\Sigma$, $\forall \alpha \in \Sigma$, telle que $\mathcal{I}_c \not\models \alpha$, on note :* $\epsilon(\alpha, \mathcal{I}_c) = \frac{1}{|\mathcal{C}_{\bar{\mathcal{I}_c}}(\Sigma)|}$.

On peut remarquer que lorsque le nombre de clauses falsifiées diminue, le degré d'appartenance d'une clause falsifiée à un MUS augmente. Une clause falsifiée α a un degré maximum ($\epsilon(\alpha, \mathcal{I}_c) = 1$), si elle est l'unique clause falsifiée par l'interprétation. Dans le cas où l'instance est insatisfiable, elle appartient effectivement à un MUS. Lorsque le nombre de clauses falsifiées augmente, ce degré d'appartenance tend vers 0. Cette mesure nous servira dans notre approche hybride afin de choisir dans un minimum local, une clause falsifiée à satisfaire (« fixer ») par l'approche CDCL.

Exemple 3.2 *Reprenons l'exemple 3.1 et considérons l'interprétation complète $\mathcal{I}_c = \{a, \bar{b}, \bar{c}, \bar{d}, e\}$ on a :*
– $\epsilon(\bar{a} \vee b, \mathcal{I}_c) = \epsilon(\bar{a} \vee c \vee d, \mathcal{I}_c) = \frac{1}{2}$

4 Travaux connexes

Comme suggéré dans l'introduction, un certain nombre d'approches ont déjà été proposées pour combiner la recherche locale avec un solveur de type DPLL. Ces hybridations peuvent être rangées suivant trois catégories différentes.
Premièrement, le solveur de base est DPLL. Dans ce cas, RL est utilisé pour aider DPLL [26, 3, 10, 16]. Toutes ces approches utilisent la recherche locale pour calculer le prochain point de choix. Certaines de ces approches tentent de concentrer la recherche sur des parties insatisfiables de la formule [26], d'autres sur la partie satisfiable [1, 10]. De plus, cette étape peut être réalisée avant la recherche [3] ou dynamiquement à chaque noeud de l'arbre de recherche [26].
La seconde catégorie d'hybridation consiste à utiliser DPLL pour venir en aide à la recherche locale [15, 21]. Dans [15], le solveur DPLL est utilisé pour trouver des dépendances entre les variables. La recherche locale est ensuite appelée sur un sous-ensemble de variables. Tandis que dans [21] (notons que cette méthode est utilisée dans le cadre de problèmes de satisfaction de contrainte), l'approche complète est utilisée pour effectuer un filtrage sur une interprétation partielle. Si celle-ci est insatisfiable une autre interprétation partielle est calculée à l'aide de la recherche locale.
Finalement, la dernière catégorie regroupe les méthodes où les deux approches coopèrent [9, 23] équitablement. La méthode proposée dans [23] étant une amélioration de la méthode proposée dans [9]. Dans ces approches, la recherche locale est utilisée pour trouver une solution. Après un temps donné, si aucune solution n'est trouvée, la recherche locale est stoppée et un solveur CDCL est lancé sur l'ensemble des clauses insatisfaites par l'interprétation courante. Si l'insatisfiabilité est prouvée sur la sous-formule, le problème est aussi prouvé comme insatisfiable.

5 Intuition

Dans cette section, nous présentons les concepts qui nous ont conduit à développer SATHYS. Tout d'abord notons que notre méthode a pour base une recherche locale, et que la partie CDCL du solveur travaille comme une liste tabou dans le cas d'instances satisfiables et essaie de se focaliser sur un MUS dans le cas d'instances insatisfiables. La suite de cette section est organisée de manière à expliquer ces deux points de vue.

5.1 Instances SAT

Beaucoup de travaux ont été menés sur les métaheuristiques. Parmi ceux-ci, la recherche tabou a été introduite en 1989 par Glover [11] et étendue au cas de SAT en 1995 [25]. La recherche tabou consiste, à partir d'une position donnée (interprétation), à en explorer le voisinage et à choisir la position dans le voisinage qui minimise la fonction objectif. Il est important de noter qu'il est possible d'atteindre une position où toutes les interprétations voisines ont une valeur plus élevée. Lorsque cela arrive, une interprétation voisine augmentant la valeur de la fonction objectif est choisie. Le risque principal est qu'à l'étape suivante, on retombe dans le même minimum local d'où l'on vient d'échapper. L'heuristique a donc besoin de mémoire. Les positions explorées sont conservées dans une file FIFO (liste tabou) d'une taille donnée, qui est un paramètre de la méthode. Cette file doit conserver des positions complètes, ce qui peut nécessiter une grande quantité d'espace mémoire. Cette difficulté peut être contournée en ne gardant que les mouvements précédents, associés à la valeur de la fonction à minimiser. La taille de la liste tabou est un paramètre important et un certain nombre de travaux ont été réalisés pour essayer d'en optimiser la longueur, statiquement [26] ou dynamiquement [2].
Dans cet article, nous proposons de gérer l'ensemble des variables tabou à l'aide d'une interprétation partielle calculée par la propagation unitaire. Quand une variable devient tabou, elle est assignée dans un solveur CDCL et la propagation unitaire est effectuée. L'interprétation partielle ainsi obtenue est alors utilisée comme liste tabou. L'avantage est double : premièrement, la taille de la liste tabou est gérée de manière dynamique, elle dépend de la propagation unitaire

et des backjumpings. Deuxièmement, la propagation unitaire permet de capturer un certain nombre de dépendances fonctionnelles.

5.2 Instances UNSAT

Tout d'abord, notons que si une instance est insatisfiable, quelle que soit l'interprétation il existe au moins une clause falsifiée. De plus, si une instance est insatisfiable, elle contient au moins un MUS. Ce MUS est bien souvent plus petit que la formule initiale et peut contenir moins de variables. Dans ce cas, il pourrait être intéressant de focaliser la recherche sur ce sous-ensemble de variables.
Dans le cadre de la détection de MUS, Grégoire *et al.* [12, 14] ont montré que la recherche locale fournissait une bonne heuristique en ce qui concerne la détection de noyaux insatisfiables. Leur approche utilise les propriétés suivantes pour isoler les clauses susceptibles d'appartenir à un MUS.

Proposition 3 *Dans un minimum local ou global, toutes les clauses falsifiées sont critiques.*

Proposition 4 *Dans un minimum local ou global, au moins une clause de chaque* MUS *est critique.*

La méthode proposée dans cet article se base sur ce principe. Quand un minimum local est atteint, la proposition 3 assure que l'ensemble des clauses falsifiées par l'interprétation partielle est critique. Puisqu'au moins une de ces clauses appartient à un MUS (proposition 4), il semble donc naturel de choisir le prochain point de choix dans l'une d'elles.

6 Implantation

Comme expliqué précédemment, notre approche SATHYS est basée sur la recherche locale. Le processus de recherche consiste à se déplacer d'une interprétation à une interprétation voisine jusqu'à obtenir une solution. À chaque étape, on essaie de réduire le nombre de clauses falsifiées (phase de descente). Lorsque aucune descente n'est possible, on se trouve dans un minimum local. Dans ce cas, il y a deux possibilités :

1. soit CDCL est appelé. Une clause falsifiée est choisie et un littéral de celle-ci est sélectionné. Ce littéral sera utilisé comme point de choix et la propagation unitaire sera effectuée ;
2. soit un critère d'échappement classique est utilisé.

Dans les deux cas, on échappe du minimum local et la partie RL de notre solveur hybride peut de nouveau être utilisée. Notons que les variables fixées par la partie CDCL ne peuvent plus être flippées par la partie RL. De plus, la partie CDCL peut conduire à un conflit. Dans ce cas, une analyse de conflit est effectuée, une clause est apprise et un backjumping est réalisé. Lorsque cela arrive, certaines variables redeviennent libres et peuvent de nouveau être flippées. L'analyse de conflit peut aussi prouver l'insatisfiabilité de la formule.

L'algorithme 3 prend en entrée une formule CNF Σ et retourne SAT ou $UNSAT$. Quatre variables sont utilisées : une interprétation complète $\mathcal{I}_c$ pour la partie recherche locale (initialisée de manière aléatoire), une interprétation partielle $\mathcal{I}_p$ pour la partie CDCL (initialisée à vide), Γ l'ensemble des clauses apprises et un vecteur ζ contenant un ensemble de couples de la forme (α, p), où α est une clause de $\Sigma \cup \Gamma$ et $p \in [0, 1]$ représentent la probabilité avec laquelle une nouvelle variable appartenant à α doit être fixée. Afin de ne pas considérer les variables fixées, la partie RL travaille sur la sous-formule $\Sigma_{|\mathcal{I}_p}$. Si l'interprétation complète courante est un modèle de $\Sigma_{|\mathcal{I}_p}$ alors SATHYS termine et retourne SAT (lignes 7-8). Sinon, s'il existe une interprétation voisine de $\mathcal{I}_c$ permettant de diminuer le nombre de clauses falsifiées, celle-ci devient l'interprétation courante (lignes 9-16). Si aucune interprétation voisine ne permet une descente, on se trouve dans un minimum local (ligne 17). Dans ce cas, la fonction $updateDegré$ (algorithme 4) est appelée et le vecteur ζ est mis à jour. Un couple $(\alpha, p) \in \zeta$ est choisi (ligne 19), tel que α est falsifiée par $\mathcal{I}_c$ et qu'il n'existe pas de tuple $(\beta, q) \in \zeta$ avec β falsifiée par $\mathcal{I}_c$ et $q > p$. Avec la probabilité p, la fonction fix (ligne 21) est appelée pour fixer de nouvelles variables et (si un conflit arrive durant la propagation unitaire) en libérer d'autres. À ce stade la partie CDCL peut montrer l'insatisfiabilité (ligne 22). Sinon, avec la probabilité $1 - p$ un critère d'échappement classique est utilisé (voir section 3.1).

Ce processus est répété un certain nombre de fois donné au départ ($MaxFlips$, ligne 6). Après quoi, tant qu'une solution n'a pas été trouvée (ou une preuve de l'insatisfiabilité de la formule) le solveur redémarre le processus précédent avec une nouvelle interprétation initiale. Comme dans le cas des solveurs CDCL classiques, la base de nogoods est nettoyée de temps à autres.

On peut noter que l'approche proposée, même si elle peut prouver la satisfiabilité et l'insatisfiabilité, est une méthode incomplète. Cependant, si l'on augmente la valeur de $MaxFlips$ à chaque itération de l'algorithme et que la base de clauses apprises n'est pas nettoyée alors l'algorithme est clairement complet.

La fonction $updateDegré$ est présentée dans l'algorithme 4. Elle prend en entrée Σ une formule CNF, Γ un ensemble de clauses apprises, $\mathcal{I}_c$ une interprétation complète, ainsi qu'un ensemble de couples (clauses, probabilité) ζ pouvant être modifié. La fonction présentée ici s'appuie sur la définition 4 pour mettre à jour les degrés d'appartenance à un MUS associés aux clauses falsifiées. Pour ce faire, la fonction parcourt l'ensemble des clauses falsifiées de $\Sigma \cup \Gamma$ et met à jour les degrés associés à chacune d'elles. La fonction termine en ayant mis à jour le vecteur ζ.

La fonction fix est présentée dans l'algorithme 5. Cette fonction travaille comme un solveur CDCL. Elle prend en entrée une clause α, ainsi qu'un ensemble de tuples ζ, l'interprétation complète $\mathcal{I}_c$ et $\mathcal{I}_p$ qui pourront être modifiées, et retourne $UNSAT$ si l'insatisfiabilité de la formule a été

Algorithme 3 : hybride

Input : Σ une formule CNF
Result : SAT si Σ est satisfiable, $UNSAT$ sinon

1 $\zeta \leftarrow \{(c,0)|c \in \Sigma\}$;
2 $\Gamma \leftarrow \emptyset$;
3 **while** *(*$true$*)* **do**
4 $\quad \mathcal{I}_c \leftarrow$ Init(Σ);
5 $\quad \mathcal{I}_p \leftarrow \emptyset$;
6 $\quad$ **for** $j \leftarrow 1$ *to* $MaxFlips$ **do**
7 $\quad\quad$ **if** $\mathcal{I}_c \models \Sigma_{|\mathcal{I}_p}$ **then**
8 $\quad\quad\quad$ **return** SAT;
$\quad\quad$ `/* ensemble de clauses falsifiées */`
9 $\quad\quad$ $\Delta = \{\alpha \in \{\Sigma \cup \Gamma\}_{|\mathcal{I}_p} | \mathcal{I}_c \not\models \alpha\}$;
10 $\quad\quad$ **while** $\Delta \neq \emptyset$ **do**
11 $\quad\quad\quad$ $\alpha \in \Delta$;
12 $\quad\quad\quad$ **if** $\exists x \in \alpha$ *permettant une descente* **then**
13 $\quad\quad\quad\quad$ $flip(x)$;
14 $\quad\quad\quad\quad$ break;
15 $\quad\quad\quad$ **else**
16 $\quad\quad\quad\quad$ $\Delta \leftarrow \Delta \setminus \{\alpha\}$;
17 $\quad\quad$ **if** $\Delta = \emptyset$ **then** `/* minimum local */`
18 $\quad\quad\quad$ updateDegré($\Sigma, \Gamma, \mathcal{I}_c, \zeta$);
$\quad\quad\quad$ `/* clause falsifiée ayant le degré max */`
19 $\quad\quad\quad$ $(\alpha, p) \in max(\{(\beta, q) \in \zeta | \ tel\ que\ \mathcal{I}_c \not\models \beta\})$;
20 $\quad\quad\quad$ **with** une probabilité p **then**
21 $\quad\quad\quad\quad$ **if** *(*$fix(\Sigma, \Gamma, \mathcal{I}_c, \mathcal{I}_p, \alpha)$=$UNSAT$*)* **then**
22 $\quad\quad\quad\quad\quad$ **return** UNSAT;
23 $\quad\quad\quad$ **else**
24 $\quad\quad\quad\quad$ utiliser un critère d'échappement classique;

Algorithme 4 : updateDegré

Input : Σ une CNF, Γ un ensemble de clauses apprises, $\mathcal{I}_c$ une interprétation complète de Σ, ζ un ensemble de couples

for $(\alpha, p) \in \zeta$ *tel que* $\alpha \in \mathcal{C}^-_{\mathcal{I}_c}(\Sigma)$ **do**
$\quad$ **if** $p < \frac{1}{|\mathcal{C}^-_{\mathcal{I}_c}(\Sigma)|}$ **then**
$\quad\quad$ $\zeta \leftarrow \{\zeta \setminus \{(\alpha, p)\}\} \cup \{(\alpha, \frac{1}{|\mathcal{C}^-_{\mathcal{I}_c}(\Sigma)|})\}$;
for $(\alpha, p) \in \zeta$ *tel que* $\alpha \in \mathcal{C}^-_{\mathcal{I}_c}(\Gamma)$ **do**
$\quad$ **if** $p < \frac{1}{|\mathcal{C}^-_{\mathcal{I}_c}(\Sigma)+1|}$ **then**
$\quad\quad$ $\zeta \leftarrow \{\zeta \setminus \{(\alpha, p)\}\} \cup \{(\alpha, \frac{1}{|\mathcal{C}^-_{\mathcal{I}_c}(\Sigma)+1|})\}$;

Algorithme 5 : fix

Input : α une clause
Output : Σ une CNF, Γ une CNF, ζ un ensemble de couples, $\mathcal{I}_c$ une interprétation complète, $\mathcal{I}_p$ une interprétation partielle
Result : $UNSAT$ si l'insatisfiabilité est prouvée, $UNKNOWN$ sinon

1 $\gamma \leftarrow \emptyset$;
2 $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup \{x\}$ tel que $\bar{x} \in \alpha$;
3 $\gamma \leftarrow propagationUnitaire(\Sigma \cup \Gamma)$;
4 **if** $\gamma \neq \emptyset$ **then**
5 $\quad$ $\beta =$analyseConflit($\Sigma \cup \Gamma$,$\mathcal{I}_p$,γ);
6 $\quad$ $bl =$calculNiveauBackjumping(γ,$\mathcal{I}_p$);
7 $\quad$ **if** *(*$bl < 0$*)* **then return** $UNSAT$;
8 $\quad$ $\Gamma \leftarrow \Gamma \cup \{\beta\}$;
9 $\quad$ $\zeta \leftarrow \zeta \cup \{(\beta, 0)\}$
10 $\rho \leftarrow \{x \in \mathcal{I}_c | \ \bar{x} \in \mathcal{I}_p\}$;
11 $\mathcal{I}_c \leftarrow \mathcal{I}_c \setminus \{\bar{x} | \ x \in \rho\} \cup \rho$;
12 **return** $UNKONWN$;

démontrée et $UNKNOWN$ sinon. Le but de cette fonction est de fixer une nouvelle variable. Pour ce faire un littéral $x \in \alpha$ est choisi pour être ajouté à l'interprétation partielle $\mathcal{I}_p$ et la propagation unitaire est effectuée (lignes 2-3). Si un conflit est atteint à la suite de la propagation unitaire, une analyse du conflit est effectuée, bl le niveau de backjumping est calculé et l'interprétation partielle est mise à jour (ligne 6). Dans le cas où le niveau de *backjump* est inférieur à zéro, la formule est prouvée insatisfiable et $UNSAT$ est retourné (ligne 7). Sinon, le *nogood* β est ajouté à la base des clauses apprises Γ (ligne 8) et le tuple $(\beta, 0)$ est ajouté à ζ (ligne 9). Pour terminer, l'interprétation complète $\mathcal{I}_c$ est mise à jour avec l'aide de l'interprétation partielle de façon à ce que toutes les variables communes à $\mathcal{I}_c$ et $\mathcal{I}_p$ aient la même valeur (notons que $\mathcal{I}_p \subseteq \mathcal{I}_c$).

7 Expérimentations

Les résultats expérimentaux reportés dans cette section ont été obtenus sur un Xeon 3.2 GHz avec 2 GByte de RAM. Le temps CPU a été limité à 1200 secondes. L'approche testée est basée sur l'algorithme 3. Dans notre cas, le critère d'échappement utilisé consiste à flipper une variable tel que le nombre de clauses falsifiées soit le plus petit possible (stratégie *best*). Notre approche a été comparée avec :

- trois méthodes de de recherche locale :
  1. WSAT [29] (stratégie best) ;
  2. RSAPS [20] ;
  3. ADAPTG2 [24] ;
- deux approches hybrides soumises à la dernière compétition SAT 2009 :
  1. HYBRIDGM [1] ;
  2. HYBRID1 [24] ;
- et deux méthodes complètes :
  1. CLS une méthode de recherche locale complète par ajout du processus de résolution [8] ;
  2. MINISAT [6] un solveur CDCL.

	Crafted		Industrial		Random	
	sat	unsat	sat	unsat	sat	unsat
ADAPTG2	326	0	232	0	1111	0
RSAPS	339	0	226	0	1071	0
WSAT	259	0	206	0	1012	0
CLS	235	75	227	102	690	0
SATHYS	340	174	473	266	930	17
HYBRIDGM	290	0	209	0	1114	0
HYBRID1	329	0	277	0	1126	0
MINISAT	402	369	588	414	609	315

TAB. 1 – Comparaison entre SATHYS et les solveurs SAT état de l'art

Les instances utilisées sont issues des dernières compétitions SAT (`www.satcompetition.org`). Elles sont divisées en trois catégories : « *crafted* » (instances académiques – 1439 instances), « *industrial* » (instances issues de problèmes réels – 1305) et « *random* » (2172). Toutes les instances sont prétraitées par SatElite [5].

Le tableau 1 résume les résultats obtenus sur un large panel d'instances. Pour plus de détail sur la partie expérimentation, le lecteur peut se référer à `http ://www.cril.fr/~lagniez/sathys`.

Pour chaque catégorie et pour chaque solveur nous reportons le nombre d'instances résolues. Le solveur MINISAT, reconnu comme l'un des meilleurs solveur SAT complet (de type CDCL) est utilisé pour mesurer le gap séparant les approches de types recherche locale et les solveurs SAT modernes sur les instances *industrial* et *crafted*. Sur les instances aléatoires satisfiables, la recherche locale est généralement meilleure que les méthodes complètes.

Avant d'analyser plus finement la table des résultats (tableau 1), remarquons que seul trois solveurs sont capables de prouver l'insatisfiabilité (MINISAT, SATHYS et CLS) et que les deux solveurs hybrides, soumis à la dernière compétition SAT, ne sont pas capables de prouver l'insatisfiabilité dans le temps imparti.

Sur les instances *crafted*, SATHYS est vraiment compétitif et résout approximativement le même nombre d'instances satisfiables que RSAPS, ADAPTG2 et les deux approches hybrides utilisées. De plus, SATHYS résout plus d'instances que WSAT sur lequel il repose. En ce qui concerne les instances insatisfiables, notre approche est moins efficace que MINISAT mais résout beaucoup plus d'instances que CLS.

Concernant les instances de la catégorie *industrial*, SATHYS résout deux fois plus d'instances satisfiables que les méthodes RL et hybrides. Comme précédemment, sur les instances insatisfiables, notre solveur est meilleur que CLS mais moins efficace que MINISAT.

Ces résultats montrent que l'analyse de conflits permet de résoudre de manière efficace les instances SAT et UNSAT.

Finalement, pour la catégorie random, nous pouvons noter que SATHYS n'est pas efficace sur les instances insatisfiables. Comme pointé par les résultats de MINISAT, l'apprentissage ne fournit pas de bons résultats sur les instances insatisfiables générées aléatoirement. En ce qui concerne les instances satisfiables, SATHYS est moins efficace que la RL et les deux approches hybrides mais résout plus d'instances que MINISAT et CLS.

Pour résumer, le solveur SATHYS est beaucoup plus efficace que les méthodes basées sur la recherche locale et les méthodes hybrides connues. Notre approche améliore WSAT sur lequel elle est basée. Même si MINISAT est le meilleur solveur sur les catégories *crafted* et *industrial*, ces premiers résultats sont très encourageants et permettent de réduire le fossé qui existe entre les solveurs à base de recherche locale et les approches complètes de type DPLL.

8 Conclusion

Dans ce papier, nous avons proposé une nouvelle approche hybride pour SAT. Notre méthode utilise la puissance des solveurs de type CDCL pour sortir des minima locaux mais aussi pour prouver l'insatisfiabilité. Nous avons également introduit une nouvelle mesure permettant d'estimer le degré d'appartenance d'une clause à un MUS. Cette mesure est ensuite intégrée dans notre approche hybride pour sélectionner une clause pertinente. Les expérimentations que nous avons conduites sur un large panel d'instances issues des dernières compétitions internationales sur SAT, montrent clairement que notre approche améliore les algorithmes de type recherche locale. Elle réduit aussi de manière substantielle le fossé en termes de performance séparant les solveurs SAT modernes des approches de type recherche locale. Nos expérimentations montrent également que notre schéma d'hybridation est plus puissant que ceux proposés et soumis récemment à la compétition SAT 2009.

Références

[1] A. Balint, M. Henn, and O. Gableske. A novel approach to combine a sls- and dpll-solver for the satisfiability problem. In *proceedings of SAT*, 2009.

[2] R. Battiti and M. Protasi. Reactive search, a history-sensitive heuristic for max-sat. *J. Exp. Algorithmics*, 2 :2, 1997.

[3] J. Crawford. Solving satisfiability problems using a combination of systematic and local search. In *Second Challenge on Satisfiability Testing organized by Center for Discrete Mathematics and Computer Science of Rutgers University*, 1996.

[4] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communication of ACM*, 5(7) :394–397, 1962.

[5] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[6] N. Een and N. Sörensson. An extensible SAT-solver. In *proceedings of SAT*, pages 502–518, 2003.

[7] E.Gregoire, B. Mazure, and C. Piette. Tracking muses and strict inconsistent covers. In *Sixth ACM/IEEE In-*

ternational Conference on Formal Methods in Computer Aided Design(FMCAD'06), pages 39–46, San Jose (USA), November 2006.

[8] H. Fang and W. Ruml. Complete local search for propositional satisfiability. In *proceedings of AAAI*, pages 161–166, 2004.

[9] L. Fang and M. Hsiao. A new hybrid solution to boost SAT solver performance. In *proceedings of DATE*, pages 1307–1313, 2007.

[10] B. Ferris and J. Fröhlich. Walksat as an informed heuristic to dpll in sat solving. Technical report, CSE 573 : Artificial Intelligence, 2004.

[11] F. Glover. Tabu search - part i. *ORSA Journal of Computing*, pages 190–206, 1989.

[12] E. Gregoire, B. Mazure, and C. Piette. Extracting MUSes. In *proceedings of ECAI*, pages 387–391, 2006.

[13] E. Gregoire, B. Mazure, and C. Piette. Boosting a complete technique to find mss and mus thanks to a local search oracle. In *International Joint Conference on Artificial Intelligence(IJCAI'07)*, pages 2300–2305, Hyderabad (India), January 2007.

[14] E. Gregoire, B. Mazure, and C. Piette. Local-search extraction of muses. *Constraints*, 12(3) :325–344, September 2007.

[15] D. Habet, C.M. Li, L. Devendeville, and M. Vasquez. A hybrid approach for sat. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, pages 172–184, 2002.

[16] W. Havens and B. Dilkina. A hybrid schema for systematic local search. In *Canadian Conference on AI*, pages 248–260, 2004.

[17] E. Hirsch and A. Kojevnikov. Unitwalk : A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematical and Artificial Intelligence*, 43(1) :91–111, 2005.

[18] H. Hoos and T. Stützle. *Stochastic Local Search : Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[19] J. Huang. The effect of restarts on the efficiency of clause learning. In *proceedings of IJCAI*, pages 2318–2323, 2007.

[20] F. Hutter, D. Tompkins, and H. Hoos. Scaling and probabilistic smoothing : Efficient dynamic local search for SAT. In *proceedings of CP*, pages 233–248, 2002.

[21] N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. In *AAAI/IAAI*, pages 169–174, 2000.

[22] Henri Kautz and Bart Selman. Ten challenges redux : Recent progress in propositional reasoning and search. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03)*, volume 2833 of *Lecture Notes in Computer Science*, pages 1–18, Kinsale, Ireland, September 2003. Springer.

[23] F. Letombe and J. Marques-Silva. Improvements to hybrid incremental sat algorithms. In *proceedings of SAT*, pages 168–181, 2008.

[24] C.M. Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for sat. In *proceedings of SAT*, pages 121–133, 2007.

[25] B. Mazure, L. Saïs, and E. Grégoire. Twsat : a new local search algorithm for sat : performance and analysis. In *Proceedngs of the Workshop CP95 on Solving Really Hard Problems*, pages 127–130, 1995.

[26] B. Mazure, L. Saïs, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Ann. Math. Artif. Intell.*, 22(3-4) :319–331, 1998.

[27] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *proceedings of DAC*, pages 530–535, 2001.

[28] B. Selman and H. Kautz. An empirical study of greedy local search for satisfiability testing. In *proceedings of AAAI*, pages 46–51, 1993.

[29] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *proceedings of AAAI*, pages 337–343, 1994.

[30] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, pages 50–54, 1997.

[31] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *proceedings of ICCAD*, pages 279–285, 2001.